MIGRATING LEGACY SKILLS-BASED SIMULATION SYSTEMS TO A DISTRIBUTED TRAINING-BASED INFRASTRUCTURE

Mark Falash
Eytan Pollak, Ph.D.
Lockheed Martin Information Systems
Orlando, Florida

ABSTRACT

Skills-based training systems tend to be standalone systems, or are networked to a small number of like platforms through reflective memory or Scramnet-like devices. Architecturally, skills-based and collective training systems share several attributes, some of which can conflict with attributes required to migrate to a distributed collective training environment. Emerging programs are rapidly moving away from expensive, proprietary hardware solutions to commercial-off-the-shelf solutions based on commodity PC-based products. This paper examines the architectural differences between existing, tightly coupled skills-based training systems and loosely coupled distributed collective training systems. In addition, the approach used to migrate an existing skills-based simulation system to a distributed infrastructure environment and the attendant lessons learned are reviewed.

ABOUT THE AUTHORS

Eytan Pollak, Ph.D., is the Synthetic Environment Product Technology Manager for Training and Simulation Solutions at Lockheed Martin Information Systems in Orlando, Florida. Projects include the development of Common Architectures for Simulation Systems, Reconfigurable Simulators, Embedded Simulation Systems, Synthetic Environment Technologies, and the integration of HLA-compliant systems. He has provided technical leadership and guidance for collective and distributed training systems for ground and aviation platforms and other research and external technology programs. He has published papers on distributed simulation, embedded simulation, reconfigurable simulators, and common architecture for simulation systems. Dr. Pollak serves as adjunct professor of Electrical and Computer Engineering at the University of Central Florida. He earned a Ph.D. from Purdue University. Facsimile 407-306-4708, telephone 407-306-3791, email: eytan.pollak@lmco.com

Mark Falash is the lead Senior Software Engineer responsible for the design and development of the initial phase of the Lockheed Martin Core Architecture for Trainers, LM-CORE, and infrastructure. Other recent projects include the integration of High Level Architecture (HLA)-compliant systems, Reconfigurable Simulators, UK-CATT, WARSIM, and software architect for the CCTT. His responsibilities included software architecture changes, performance modeling and analysis, and rehosting proprietary solutions to commercial-of-the-shelf COTS products including uni-processors and symmetric multi-processors (SMPs). He earned an MS in Computer Science from California State University. Facsimile 407-306-4708, telephone 407-306-5267, email: mark.falash@lmco.com

MIGRATING LEGACY SKILLS-BASED SIMULATION SYSTEMS TO A DISTRIBUTED TRAINING-BASED INFRASTRUCTURE

Mark Falash
Eytan Pollak, Ph.D.
Lockheed Martin Information Systems
Orlando, Florida

Introduction

Most early (and existing) simulation-based training systems are used to teach a trainee to drive or fly a vehicle. These systems focus on operational skills and vehicle procedures, such as the proper sequence of actions to start or shutdown the vehicle, fire weapons, landing or take-off procedures, etc. Skills-based training systems tend to be standalone systems, or are networked to a small number of like platforms through reflective memory or Scramnet-like devices. Architecturally, skills-based and collective training systems share several attributes. These architectural attributes can conflict with attributes required to migrate to a fully distributed collective training environment. In addition, emerging programs are rapidly moving away from expensive – and often proprietary – hardware solutions to commercial-off-the-shelf (COTS) solutions based on commodity PC-based products. This paper examines the architectural differences between existing, tightly coupled skills-based training systems and loosely coupled distributed collective training systems. In addition, the approach used to migrate two existing skills-based simulation systems to a distributed infrastructure environment and the attendant lessons learned are reviewed.

Driving Requirements

Traditionally, skills-based training systems have been driven by a single overarching requirement: to train students in the procedures and skills necessary to operate a specified vehicle and its onboard equipment, such as navigation or weapon systems. Collective training systems, on the other hand, have been primarily driven by the need to train students on doctrine, tactics, and decision-making within task organizations, ranging from platoons to battalions. It is these differences in driving requirements that have led to the differing architectures and design constraints encountered in regard to these respective systems.

Traditional Skills-Based Training Notional Architecture Overview

As a result of the objective to train skills directly related to the operation of the vehicle and its equipment, skills-based simulator systems evolved to have very specific characteristics (Figure 1).

1. High fidelity crew stations that closely replicate real equipment, and often integrate actual vehicle equipment with the simulator itself.
2. Control loading and motion platforms to simulate vehicle force feedbacks and motions.
3. Instructor operator stations (IOS) tailored to and connected directly to a single simulator.
4. A tactical environment that simulates targets, emissions, and weather conditions; this environment can at times simulate elements associated with the vehicle, such as weapons fly-out.
5. Visual systems used to provide out-the-window (OTW) imagery manage, visualize and operate (Mission Functions) on natural environment and tactical environment objects.
6. Tightly coupled components based on specialized devices with high throughput such as Scramnet or reflective memory.

Due to the focus on skills training, these systems have primarily been operated as standalone systems, with exercises developed to test, collect, and analyze data regarding operational procedures (i.e., take-off, landing, weapon firing, etc). Status of vehicle equipment,

position of switches, and student actions are readily available to the IOS and are collected for

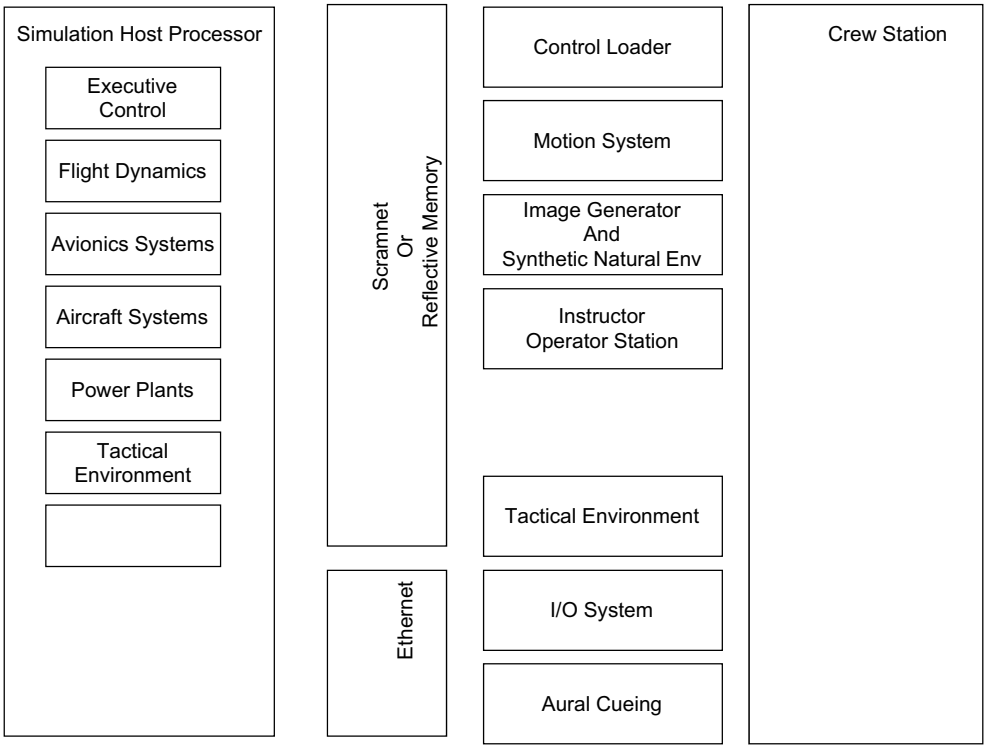

Figure 1. Traditional Skills Based Training Architecture

use during the After Action Review (AAR). AAR sessions are conducted inside the simulator with the Instructor and student in the positions used during the exercise. Exercises contain relatively few platforms or targets, whether they are friendly or enemy forces. The visual system, it turns out, is often the component that the simulator is designed around, creating what can be termed image generator (IG)-centric simulators. The IG provides not only the visual representations required, but also provides critical mission function capabilities (i.e., height above terrain (HAT), line-of-sight (LOS) collision detection, etc.). The IOS operator can control almost any aspect of the simulator from the IOS station. The IOS operators can position targets and the ownship, set specific weather conditions, insert malfunctions, and even fly the craft.

Based solely on the components required to develop a manned simulator, or on most any component taken in isolation, one might see few differences between skills-based training systems and collective training systems beyond the method by which the various components are interconnected; however, as a whole the software capabilities for the infrastructure are unique in several aspects, whereas vehicle (crew station) hardware components are driven by differences in fidelity. By first considering the number of platforms required to meet training objectives and the interconnection (Scramnet, reflective memory, or shared memory) used between the tactical environment, the Host Simulation, and the IG, we begin to understand important design constraints that will distinguish skills-based trainers from collective trainers. Image generators are computation-intensive systems that until recently required very expensive proprietary hardware and software to meet program requirements. A number of factors affect IG performance, such as update rate, number of polygons, field-of-view (FOV). and number of mission functions or animations being processed. In general, the number of platforms required to support an exercise is fewer than 40 (with 50 platforms representing a worst-case scenario), which often drives the polygon count along with the level of detail (LOD) of the terrain. Given the state-of-the-art of host computers, tactical environment

processors, and image generators the system throughput requirements do not exceed that capacity of any part. The tactical environment models and updates the position, appearance, etc., of all 50 platforms in global shared memory at 60Hz. This is true whether the tactical environment is executing in the same host or a different host since Scramnet or reflective memory is utilized. The vehicle simulation and IG simply pull the data from this global shared memory pool as needed – real-time data management requirements in the host are minimal or non-existent. The IG has the capacity to handle all the platforms and environment conditions modeled by the tactical environment and the ownship OTW and sensors. Previously, we coined the term IG-centric. In this architecture the IG provides the OTW and vehicle sensor capabilities as well as the representation of the environment outside the skin of the aircraft. In other words, the IG is the synthetic natural environment (SNE) and the environment is not shared with other simulation systems.

Vehicle dynamics, control loading, and motion platforms exhibit operate similarly in that ownship forces are updated in global shared memory for control loader and motion system reactions. The control loader responds by updating global shared memory with feedback resulting in vehicle dynamics. The high throughput and low latency of the global shared memory provides a tightly coupled interface that avoids latency issues associated with other more loosely coupled designs utilizing Ethernet.

Traditional Distributed Collection Training Architecture Overview

As noted previously, collective training systems are driven by the objective of training groups in doctrine, tactics, and decision-making skills within a task organization. It is presumed that the trainees already know how to operate the equipment used during the course of an exercise. Following are attributes associated with most distributed collective training systems (Figure 2).

1. Lower fidelity and reconfigurable crew stations that replicate enough of the crew station to accomplish the training tasks.
2. Control loading and motion platforms to simulate vehicle force feedbacks and motion are often not required.
3. A general-purpose exercise planning and control workstation is attached to the network in lieu of a specialized IOS.
4. Distributed Computer generated forces (CGF) are used to generate targets and emissions in lieu of tightly couple target generators. Weather server and/or models are used to generate natural environment conditions.
5. Visual system is used to provide the OTW and sensor imaging. The management of the natural environment, tactical environment objects and mission functions are executed on the host using correlated databases.
6. Loosely coupled components communicate state data via a simulation network using Distributed Interactive Simulation (DIS) or High Level Architecture (HLA) protocols and standards.

As with the previous skills-based training discussion, let us start with the number of platforms necessary to meet the training objectives of collective training systems. Two distributed training systems developed recently, the Close Combat Tactical Trainer (CCTT) and UK Combined Arms Tactical Trainer (UKCATT), were designed for platoon- and company-level training with some limited battalion behaviors, and require more than 750 platforms. Other aviation programs that utilize distributed simulation technology and involve a dense threat environment may require more than 700 entities to execute tactical training exercises. The size of these exercises will easily overwhelm any architecture or design if performance requirements are not properly allocated to appropriate system components. In addition to the larger exercises, the addition of crew or collective training requires multiple man-in-the-loop trainers that must interact during the course of the exercise. Assuming that state-of-the-art IGs are still limited to between 100 and 150 platforms at 30Hz or 60Hz, the system will need conscientious data management to meet the requirements of the large exercises and not overload any system component, such as an IG or network interface. Assuming that all platform

data is available to be displayed, some method is required to reduce the data to a level that will not overload the IG. At first glance, this might appear to be a simple problem, but in practice it is quite complex. Consider that OTW and

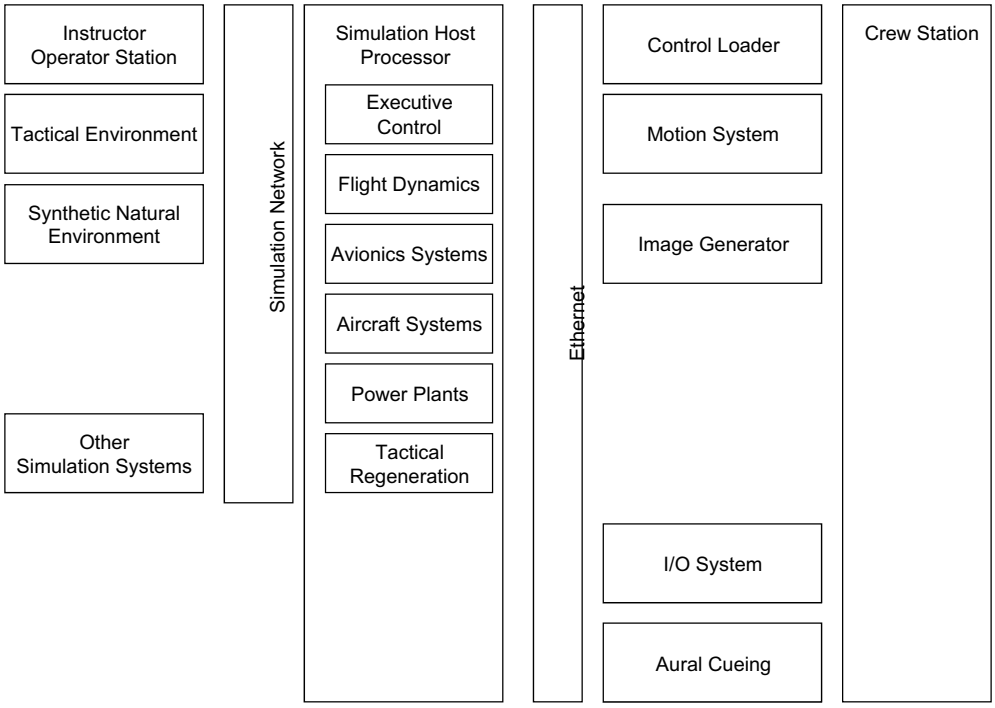

Figure 2. Traditional Collective Training Simulator Architecture

sensors can have dramatically different ranges and may be oriented in different directions – each having an entirely different FOV. In addition, simulating aircraft that travel at high rates of speed mean defining targets that may move into or out of the vehicle OTW or sensor FOV very quickly; and creating and deleting models on IGs tends to be very expensive. Even if the set of displayed platforms were optimized to those platforms within the OTW or sensors' fields of view, it would not be prudent to assume the resulting set could be processed without overloading the system. With multiple man-in-the-loop trainers along with CGF and workstations all attached to a simulation network, sending data as it changes at 30Hz or 60Hz will quickly saturate the simulation network. In order to manage the network load the DIS standard specifies use of threshold checks and dead reckoning algorithms to extrapolate position and orientation of platforms.

Architecture Comparison

Both skills-based and distributed collective training systems leverage virtual simulation technologies to achieve their training goals. Although the same technologies are utilized, the requirements that were derived from their training objects have lead to two different architectures (Table 1). With the desire to reduce the cost of simulation systems by using COTS hardware and software, and the need to expand the skills-based system to support Collective Task Training, emerging programs that seek to use their assets more efficiently are combining these two unique architectures.

Characteristic	Skills Based Systems	Collective Training Systems
Crew Station Fidelity	High	Varies
Network	Tightly coupled, standalone	Loosely coupled simulation network
Image Generation System	IG centric	Network centric
SNE Databases	IG, Radar	IG, Radar, Host/SAF
Data Management	Minimal	Large # of platforms, emitters, etc.

Table 1. Skills-Based versus Collective Training Systems

Migration of Helicopter Skills-Based Trainer

An experiment was undertaken to migrate a full flight skills-based helicopter trainer system to a distributed collective training environment. A CH53E helicopter software package was selected, in large part due to the availability of the software and the perceived need of the aircraft on emerging programs. Like most simulations of this type, the CH53E was made up of:

1. Executive control for controlling the execution of the simulation on the host platform, rate of execution, control flow, and modes.
2. Vehicle-unique software and data files that simulate vehicle dynamics and associated subsystems.
3. A tactical environment that provides modeling of targets, emitters, and weather conditions.
4. Training, monitoring, and evaluation that provide performance and evaluation data used to score or assess student performance and/or progress.
5. Interfaces between the vehicle and external subsystems, i.e., image generator, aural cues, motion system, control loader, and crew station inputs and outputs.

The primary objectives of this task were to gain hands-on knowledge and experience with migrating a legacy trainer to a distributed collective training environment while leveraging low-cost commercial products, such as a PC host and a PC IG. Due to resource constraints, CH53E functionality would not be fully realized during this initial effort since a full-up cockpit, motion system, and I/O systems were not at our disposal (Figure 2). As will be discussed later, we used other low-cost products on hand, along with some scripted inputs to mimic the vehicle's start-up procedures. The overall approach taken to migrate the CH53E simulation to a fully distributed collective training system contained the following guidelines:

1. Avoid modification of the vehicle's internal components unless absolutely necessary. Our rational was that we were using a working and verified simulation model and the objective was to update the infrastructure that manages and represents the environment that exists outside the skin of the aircraft, not the aircraft models.
2. Interface subsystems to the aircraft in a manner that avoids device or implementation dependencies. The rational for this approach was threefold: a) it was our desire to start with available resources in place of actual equipment; b) the more easily we could insert new technologies, the more easily we could leverage the price/performance curve to manage cost; and, c) the more accommodating the system was when integrating new or alternative devices, the more useful the device would become for future projects.
3. Use of well-defined Application Programming Interfaces (APIs) to interface the aircraft systems to the surrounding environment; the rational being that APIs would facilitate accomplishments of the prior two items, and APIs would allow for incremental implementation of capabilities.

Initially, we started with only the host application software source code, a design document, and none of the original hardware. The initial effort consisted of simply attempting to get the source code (Ada) to compile. The original host processor was a *Harris Nighthawk,* which was not available to the team. The source code was

originally recompiled on AIX PowerPC workstations, primarily because the infrastructure to be used existed on AIX and this platform was readily available. Shortly thereafter the source code was easily ported to the desired PC platforms running Redhat Linux. The software tree was slightly restructured, leaving vehicle unique components alone. The resulting vehicle-unique elements, along with the vehicle data collector, a new component, are as depicted (Figure 3). The Vehicle Data Collector component is dedicated to the collection of data created by the CH53E vehicle-unique components, and interfaces those vehicle-unique components with the world outside the aircraft skin. Note that the Data Collector and interfaces in actuality are a set of packages managing a diverse variety of information, i.e., Vehicle position, orientation to weapons,

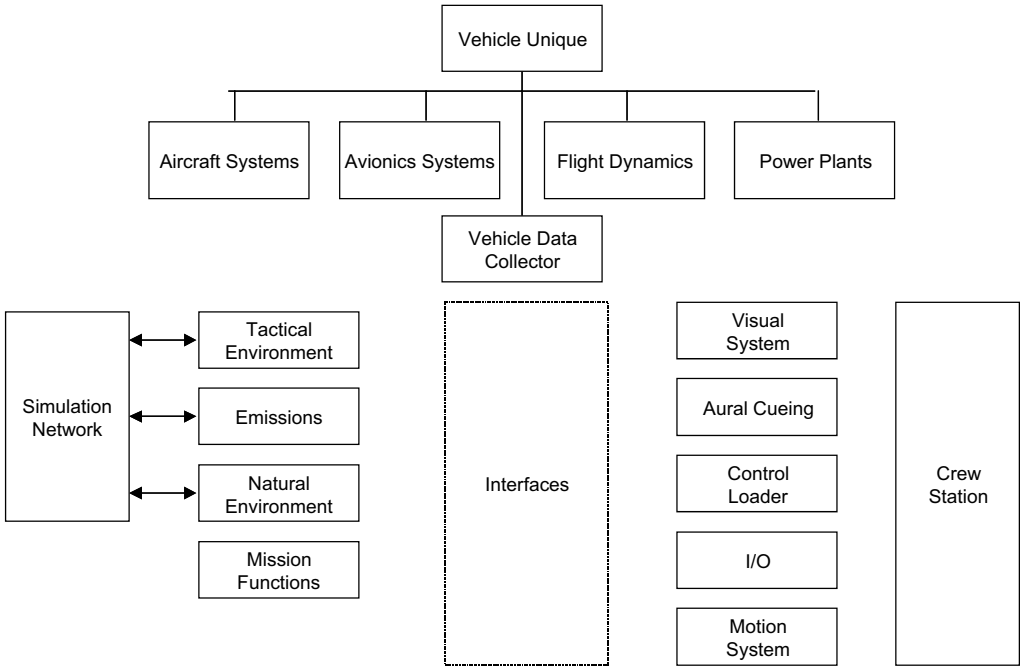

Figure 3. CH53E Software Tree

emitters, etc. Given this approach, the vehicle-unique software remains unchanged while allowing us to decouple the infrastructure, which will in turn support the insertion of components designed to manage and communicate data in a distributed environment. The configuration used for initial demonstrations included PC Host Platforms running RedHat Linux version 6.2 (Figure 4). The PC Host contains 2 Pentium III 733mHz processors and 256MB RDRAM. The top portion of the diagram depicts the vehicle-unique software, while the lower portion of the diagram illustrates the infrastructure components used to managed the distributed environment around the vehicle:

1. Process manager sets up the environment and starts the processes that make up the executing simulation. The Process Manager also monitors the simulation's health.
2. Tactical environment manages both local and remotely generated platforms that are participating in the simulation exercise. Coordinate Conversions, Dead

Reckoning, Smoothing, and Threshold checks are performed by this component.

3. Natural environment manages the models and presentations of the natural environment, ranging from pressure, wind and temperature to clouds, smoke and haze, sea states, etc.
4. Mission functions provide functions that were previously provided by the IG. These include height above terrain (HAT), line of sight LOS, etc.
5. Visual interface provides an abstract interface to the IG. This interface provides a set of APIs used to control the IG, creates, updates and deletes platforms, and manipulates environment representations such as clouds, haze, etc.
6. Crew station interfaces are used to interface the host software with the crew station components. This includes analog and discrete I/O, control loader and instruments.
7. Network is used to send and receive data from the simulation network, be it DIS or HLA.
8. Environment process is responsible for managing the dynamic environment that is not vehicle specific. For example, the environment process creates, updates environment representation on the IG using the visual interface APIs. Issues Surrounding Emerging Architectures

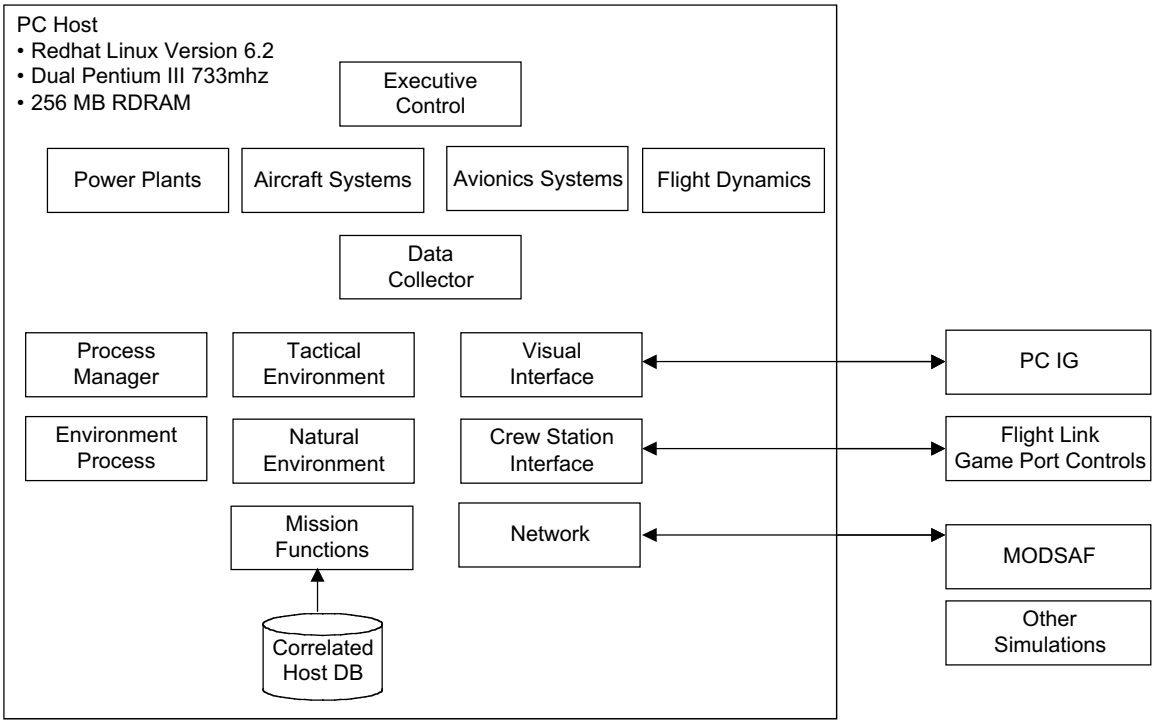

Figure 4. CH53E Configuration Used to Demonstrate Distributed Environment

So far we have alluded to some fundamental differences between the traditional skills-based training system and a distributed collective training system, as was depicted in Table 1. The industry today has significant experience with both tightly coupled standalone training and with distributed collective training systems. Future systems will combine the ability to train skills and participate is collective training exercises. The challenge will be how to merge these two different notional architectures into one that satisfies both sets of requirements. With consideration to our selection of a CH53 simulation, which architecturally fits our description of a traditional skills-based trainer (Figure 1), let us review the differences based on characteristics outlined in Table 1. Exercise size is the first element listed in the table.

Size of Exercises

We have characterized skills-based exercises as small and collective training exercises as large. Most skills-based training exercises tend to involve fewer than 40 or 50 platforms, while collective training exercises tend to be anywhere from 250 to 3000 platforms. Using a tightly coupled architecture as described, the tactical environment simply updates the global data structures at 60Hz and any interested component queries the data as needed. Since each component, including the host processor, IG, and aural cueing system can easily handle this many platforms (and corresponding updates), there is no need to manage or filter the data in order to prevent overloading any of the system components. However, when we pump it up a notch or two with exercises of 250 platforms or more, the load from platforms will overload the system if precautions are not taken. For the sake of this argument let us assume that the tactical environment can generate the necessary number of platforms, and writing to and reading from some global data area is not an issue. Given the state-of-the-art of image generators, today's IGs can handle substantially more of a load, but not enough that without some filtering of data the IG will overload attempting to process all of the platforms. This overloading issue applies to the simulation network and host processors. In a distributed collective training environment it is most likely that an Ethernet-based local area network (LAN) is used for the simulation network. Today, 100MB Ethernets are standard offerings. If attempting to update the simulation network for all 250-plus platforms every frame, there would be 15,000 updates per second – with an equal number of interrupts on the receiving processor, assuming no packet fragmentation. The DIS working groups realized this long ago; this is why DIS specifies dead reckoning and threshold checking be used to extrapolate the position of remotely generated platforms, and also specifies when to transmit locally generated platforms.

Cockpit Fidelity

Cockpit fidelity is less an architectural issue than a training requirements issue. Skills-based trainers are aimed at training a person to operate a vehicle; thus the more precisely the crew station replicates the actual vehicle the better. With collective training systems, it is already presumed that the crew knows how to operate the vehicle and less than complete functionality is generally acceptable. The training equipment will determine what is and is not required.

Networks

For the most part we can partition the networks into two types, with the first being the network that connects the simulator components. This network connects the Host processor to the various subsystems such as the IG, aural cueing, control loader, motion system, and IOS. The second is the simulation network, or the LAN that connects the simulators or simulation systems together. For skills-based trainers, many components are connected via Scramnet or reflective memory. This method of interconnection resembles shared memory, provides very high throughput, and is more expensive. This type of tightly coupled system generally does not have a simulation network for connecting to other simulators or simulations. With collective training systems, Ethernet or the equivalent is generally used to interconnect various subsystems, although Scramnet or Reflective memory could be used as well. Since collective training systems are based on many simulators or simulations interacting over a network, a separate simulation network always exists that is separate from the simulator's internal network. It is really the absence or existence of the simulation network that begins to differentiate the systems. Skills-based training systems rarely account for sharing of simulation data between simulators, whereas collective training systems are based on this premise. Another way to view this concept is to think of a skills-based trainer as a system and collective training as a system of systems.

Image Generation System

We have already touched on the role of IGs in these systems. Skills-based training systems were termed IG-centric, due to the fact that the IG provides and fully contains the synthetic natural environment. This includes mission functions to calculate LOS, HAT, perform collision detection, etc. Historically, IGs have been large, expensive proprietary systems. A technology trend is clearly well underway to

replace the proprietary hardware with low-cost commercial products. Migrating operations other than those that relate to rendering, such as mission functions, will migrate from the IG to the host processor. This not only significantly reduces the latency between the request and the result, but also facilitates interoperability: the mission functions no longer rely on the IG, therefore the simulator and non-IG components such as the CGF and workstation could eventually make use of the same algorithms and data.

Synthetic Natural Environment

The synthetic natural environment (SNE) has been fully encapsulated, owned, and managed by the IG when it comes to skills-based trainers. With collective training systems, the SNE is distributed across many simulators and each simulator (including CGF and workstations), locally manages the modeled objects, and communicates the state changes over the simulation network. Each simulator thus contains a collection of the current state of the gaming area and exercise. Since there is no single source of the SNE, each simulator creates and maintains it's own perception of the SNE. This creates several interoperability issues. First and foremost is the correlation of terrain skin and terrain features. By now most of us have seen platforms floating in the air or underground, obviously due to differences in SNE databases built from different source data and tools. Traditional methods of creating correlated databases involve processing the visual database to extract and reformat data into a correlated database. SEDRIS is a step in the right direction, were the database format and attributes is not depended on the visual system and it provides all the attributed necessary for all the training subsystems like the CGF, sensors, EW and etc.

Data Management

Historically, simulation-based training systems have not paid much attention to data management or even the data architecture aspect of these systems. With a standalone IG-centric system, data management was minimal or non-existent at the system level and was the responsibility of each subsystem. With distributed collective training, simulations are required to maintain the state of the battlefield by publishing their state to other network-attached simulations, and to collect states from platforms. As the fidelity and capabilities of the training systems – as well as the vehicles – rises, the need to manage data increases. Current implementations to electronic warfare (EW) will quickly overwhelm both networks and receiving simulations without some method of local regeneration. Several hundred platforms with an average of 1 to 10 sensors per platform will generate far more data than current entity state data. This problem will only become more difficult as the number and complexity of sensors on new and upgraded vehicles increases. Similar issues exist with the SNE: today's models are fairly simplistic, especially within the distributed training environment. As weather and atmosphere models become more realistic, additional stress on the network will be incurred along with host processing demands to extrapolate conditions locally. If for no other reasons than performance and scalability (other incentives include interoperability and fair fight) the simulation community will be driven to some common approach (such as DIS definition of dead reckoning and threshold checks) in these areas, either formally via working groups or standards.

Summary

The historical architecture or design differences between the two simulation based training systems will slowly disappear over time as legacy systems are upgraded and newer systems become operational. As we have shown to some degree with the CH53E simulator, migrating standalone skills-based training systems to an infrastructure designed for distributed collective training is technically feasible. However, issues remain regarding scalability, interoperability, and fair fight. Increasing fidelity requirements for sensors and OTW will continue in addition to increase modeling fidelities such as the SNE are literally the same issue as the migration to distributed trainer put into a different context.